Artificial Intelligence at OSU

Thomas G. Dietterich
Bruce D'Ambrosio
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

Abstract

This report describes of current research in Artificial Intelligence at Oregon State University. The five areas of active research are (a) intelligent aids for mechanical engineering design, (b) active experimentation as a method in machine learning, (c) techniques for combining logic programming and assumption-based truth maintenance, (d) methods for combining symbolic and numeric approachs to reasoning under uncertainty, and (e) problem-solving strategies for working with multiple qualitative models.

1 Design Histories as an Aid to Redesign in Mechanical Engineering—Thomas G. Dietterich and David G. Ullman[1]

1.1 Introduction

The goal of our current research is to improve the productivity of designers in mechanical engineering. Our work has focused primarily on understanding the design process and on developing tools to improve that process.

Under a grant from NSF (DMC-8514949) we have conducted a protocol-analysis study of the mechanical design process. In this study, we developed two real-world mechanical design problems. The first problem involves designing a battery-contact subassembly for a portable personal computer. The second problem involves designing a piece of manufacturing equipment (called a "flipper-dipper") that assists a worker in coating both sides of a rectangular aluminum plate with a thin film. The first problem requires knowledge of metal spring electrical contacts, injection-molded plastics, and robot assembly methods. It emphasizes the design of a high-volume commercial product. The second problem, in contrast, requires knowledge of kinematics and simple actuator technology (hydraulic, pneumatic, electro-mechanical). It emphasizes the design of low-volume "one-off" machines.

We gathered 6-10 hours of protocol data from each of six designers (three on each problem). Of these subjects, two were relative novices (one on each problem) and the remainder were experienced industrial design engineers. This large body of data has been reduced to yield detailed traces of the design steps of each subject. We are still in the process of further analyzing the data to identify the domain knowledge and control strategies of the subjects.

1.2 Current Research

Based on this study, we are currently constructing and testing a tool for aiding the design process. Rather than attacking the entire mechanical design problem, we are focusing on the area of redesign. A large fraction of the design time for a product is spent redesigning the product as specifications and manufacturing constraints change. Our hypothesis is that redesign can be substantially aided by giving the designer access to a "design history browsing tool"—a computer system that provides a convenient, graphical display of the history of the previous design.

A design history is a record of the design process that provides information on how and why each design decision was made. One of the major shortcomings of current design practice is that no record is kept of the *process* by which a design is constructed. In mechanical engineering, it is usually the case that only the final detailed drawings (and possibly some handwritten notes in the designer's notebook) are retained to document the design. The entire process by which the design was developed—the alternatives considered, the reasons for rejecting them, the tradeoffs made, and so forth—are lost. Regrettably, this is precisely the kind of information that is needed by a designer who must alter or redesign the product at some later time.

There are five research questions concerning the design history that need investigation: (a) what information should be included in a design history, (b) how should that information be collected, (c) how should that information be represented in the computer, (d) how should the information be replayed during redesign, and (e) how effective is the design history for improving redesign?

Some of these questions have already been explored by other researchers in artificial intelligence and software engineering. An important approach to software design automation is called *transformational programming* (Partsch & Steinbruggen, 1983; Green, et al., 1983). Programs

[1]Dr. Ullman is an Associate Professor of Mechanical Engineering

are developed by first writing a very high-level formal specification and then applying a series of *correctness-preserving program transformations* to refine the specification into an efficient implementation. Each program transformation is a rule that transforms a statement in a high-level language (e.g., a specification language) into an equivalent statement in a lower-level language (e.g, an implementation language).

Within this framework, a design history is a record of the transformations that were applied by the programmer (Wile, 1983; Neches, Swartout, & Moore, 1985). The record may also include the reasons for selecting each transformation (Fickas, 1985). The key idea is that during redesign, this design history can be *replayed* to transform the changed specification into a changed implementation. Of course some of the original transformations will no longer be appropriate, and at such points, the programmer must intervene and select new transformations. However, the benefits of the approach are clear: the redesign involves minimal changes to the program and the changes are guaranteed to be correct (because the transformations preserve correctness).

This work in transformational programming suggests the following strategy for mechanical CAD: (i) construct a transformational development system for automating the design process; (ii) use that system to collect, represent, and replay the transformational design history. We have decided, however, not to take this approach for two reasons. First, although it answers questions (b), (c), and (d), it does not address questions (a)—what belongs in the design history—or (e)—the effectiveness of design histories—which are the most important. Second, it would be a very difficult undertaking in mechanical engineering, because we lack formal languages for describing specifications and partial designs. It was the development of such languages for computer programs that enabled the work on transformational programming.

We are instead conducting a series of experiments aimed at questions (a), (c) and (e). Instead of *recording* design histories, we are constructing them by hand, based on the data gathered from our previous protocol-analysis experiments. Unlike the software studies, our design histories will not be replayed automatically by computer. Instead, we are developing a computer-based browsing tool to help a design engineer examine and query the design history. Supported by this tool, the design history will become one more source of information, in addition to catalogs and drawings, for assisting in the redesign process. We plan to evaluate the effectiveness of this tool through a series of controlled experiments.

This strategy has three primary advantages. First, it is feasible to perform in the near term. Second, it will provide a relatively inexpensive test of the value of design histories (compared to the construction of an automated design system). Third, it provides a test case for developing and refining formal languages for describing mechanical design specifications, implementations, and the design process itself. Hence, it addresses problems that are critical to the development of a whole range of intelligent design aids. In particular, it will provide the representational base for the development of CAD tools that can automatically record the history information while assisting the designer during the design process.

There are four basic tasks in our research plan.

Task 1: Develop a design history based on existing protocol data. Under our current NSF grant, we have already gathered the protocol data needed for this task. In addition, we have conducted very detailed analysis of selected sections of the protocol data. However, to complete this task, we need to select one of our six protocols and perform additional, detailed analysis of that protocol to construct a complete design history.

Task 2: Construct a formal representation language for design histories. As a part of our current NSF grant, we have made some progress toward the development of formal languages for representing the form and function of a design as it progresses from abstract specification to detailed implementation. However, the languages that we have developed are intended only for representing small segments of the protocol data. In this task, we will extend these representations so that they are capable of capturing the entire design history for the selected design problem.

Task 3: Construct a computer-based tool for browsing and querying the design history. The form in which the design history is made available to the designer is critical to the success of the research. The key cognitive task for an engineer facing a redesign problem is to understand and internalize the constraints inherited from the previous design that limit the possibilities for redesign. Hence, we are developing a computer-based browsing tool for examining and querying the formal design history. We plan to cross-index each design decision and design constraint with each component and function in the design. Using a mouse and window-based user interface, the engineer will be able to point at a component of the design and determine all of the constraints and design decisions that affect that component. Similarly, the engineer will be able to select any of the original specifications or any design decision and obtain a visual presentation of the impact of that specification or decision on the design.

The design history will also be available as a goal/sub-goal tree display on the screen. Pointing at any node of the tree and pressing a button will reveal the details of the design decision at that node. Shapes, colors, and related techniques will be employed to encode different kinds of design steps. The browsing tool will also support editing of the design history. Hence, it will also be used for initial entry of the design history into the computer.

Task 4: Evaluate the effectiveness of the design history To evaluate the benefits of a design history, we plan to conduct a series of experiments in which several engineers will re-design altered versions of the flipper-dipper. Two variables will be manipulated: the access to the design history and the degree of change in the specifications. Three levels of design history access will be provided: (a) access to the browsing tool described above, (b) access to a written narrative of the design history, and (c) no history information. All levels will include access to the final drawings of the original design. Two different specifications will be given. One group will be given a redesign problem in which only minor changes to the specification have been made. The other group will be asked to redesign based on a major change in the specification. The effectiveness of the designers will be measured by taking verbal and video protocols and rating the quality of the designs, the number of components changed, and the number of constraints violated.

We expect that access to the design history will help designers work faster and will lead to redesigns that change relatively few components and do not violate any constraints. However, there may be an adverse effect with major specification changes. In those cases, our subjects may become preoccupied with trying to save the original design rather than developing significantly different approaches to the problem.

1.3 Concluding Remarks

On the basis of these four tasks, it should be possible to make an accurate assessment of the utility of design history tools for aiding the redesign process. The tasks will also provide the foundation for developing industrial-level design history tools, both in terms of knowledge representation languages and user interface techniques.

2 Combining Logic Programming and Assumption-Based Truth Maintenance—Thomas G. Dietterich

2.1 Previous work on FORLOG

Two important techniques—logic programming (Clocksin & Mellish, 1984) and assumption-based truth maintenance systems (deKleer, 1986a; 1986c)—have been developed in the past few years.

Logic programming is a programming style in which programs are written declaratively as a collection of axioms and executed by an interpreter that respects the declarative semantics of the axioms. The main advantage of this approach is that the correctness of the program can be determined (at least in part) by considering the declarative semantics of the axioms. A second advantage of logic programming is that it provides a new primitive data structure—the logical (or un-bound) variable. This facilitates programming with partially-instantiated structures.

Truth maintenance systems (TMSs) can be viewed as databases that record not only the *current state* of program execution but also the justification (or history) for how the state was derived. For programs that conduct searches, this information can be used to determine the search choices that led to a contradiction so that backtracking can be performed intelligently. The assumption-based truth maintenance system (ATMS) is a particularly efficient TMS that is also very flexible. In particular, unlike previous systems, it can represent several different states (or contexts) of problem solving simultaneously, so that it can be used by search programs that employ search strategies other than simple depth-first search. Furthermore, information can be shared among the different states so that no information is stored redundantly.

Both logic programming and truth maintenance systems share the view that problem solving involves deriving new facts from old ones. Hence, it is natural to attempt to combine the two techniques. This is not entirely straightforward, because the ATMS assumes a forward-reasoning architecture while current logic programming systems use a backward-reasoning control structure. We have developed a method (see Flann, Dietterich & Corpron, 1987) for converting backward-chaining logic programs into semantically well-founded forward-chaining programs. The basic idea is to use the "forward-half" of the completion of the logic program (Clark, 1978). Consider the well-known Prolog `append` program:

```
append([],Z,Z).
append([X1|XR],Y,[X1|ZR]) :- append(XR,Y,ZR).
```

This program says that we can prove that `Z` is the result of appending the lists `X` and `Y` either by proving that `X` is the empty list and that `Y=Z` or by proving that `X` and `Z` have the same first element and that the rest of list `X`, when appended to `Y` yields the rest of list `Z`.

The equivalent FORLOG program is

$$\begin{aligned} \forall\, x,y,z\ \mathit{append}(x,y,z) \quad \supset \quad & (x = [\,] \ \wedge\ y = z) \vee \\ & \exists\, x_1, x_r, z_r\ x = [x_1|x_r] \ \wedge\ z = [x_1|z_r] \ \wedge\ \mathit{append}(x_r, y, z_r) \end{aligned} \qquad (1)$$

This states that if $append(x, y, z)$ is true, then it must be the case that either x is the empty list and $y = z$ or else x is a list whose first element is x_1, z is a list whose first element is also x_1, and the result of appending the rest of x (x_r) with y is the rest of z (z_r).

To execute the Prolog program, we set a goal that contains unbound variables and attempt to find a proof for an instance of that goal. To execute the FORLOG program, we make an assertion that contains Skolem constants and explore the consequences of that assertion. Among the consequences will be equality assertions involving the Skolem constants. For example, if we assert

$append([1],[2,3],sk_1)$ (where sk_1 is a Skolem constant), we will eventually infer that $sk_1 = [1,2,3]$ as a consequence of this assertion and rule (1).

To manage the execution of the forward-chaining logic programs, we have adopted the "consumer architecture" invented by deKleer (1986c). To improve speed, we compile the forward-chaining rules into a data-flow network. Some of the nodes in the network correspond to RETE "AND" nodes (Forgy, 1982). Other nodes correspond to "READ" and "WRITE" unifications in the Warren Abstract Machine (Warren, 1983). The result is the FORLOG automated reasoning system.

FORLOG combines all of the advantages of Prolog with the search flexibility, intelligent backtracking, and default reasoning capabilities of the ATMS. In fact, we have proved that a subset of FORLOG is procedurally isomorphic to (i.e., yields the same run-time behavior as) pure Prolog.

We believe that FORLOG also provides improved support for an extension to the logic programming paradigm that we call *programming with assertions*. In this extended style, the logic program is able to manipulate assertions without resorting to the use of `assert` and `retract` that would be necessary in Prolog.

2.2 Current Research

There are four tasks in FORLOG that require further research.

Task 1: Translate FORLOG to Commonlisp. The first task is to translate the FORLOG system, which is now written in Interlisp-D, into Commonlisp. Commonlisp is rapidly becoming the standard lisp, available on a wide variety of machines. We have received many requests for copies of FORLOG, but because it is available only in Interlisp-D (which runs only on Xerox 1100-series computers), we have been unable to honor these requests.

Task 2: Experiment with the style of programming with assertions. Prolog (and the subset of FORLOG illustrated by the *append* example) supports one basic data structuring mechanism: terms. Terms are hierarchical nestings of simple tuples, and consequently, they are unable to represent complex, non-hierarchical structures in any direct way. The natural solution to this problem is to describe complex structures using collections of assertions. We call this style of programming *programming with assertions*. Prolog provides only primitive, non-logical support for programming with assertions by the use of `assert` and `retract`. FORLOG, because of its forward chaining architecture, can be viewed as deriving new assertions from old ones—not just new variable bindings from old ones in the Prolog fashion.

We have written several small programs in this new programming style, but we need to gain more experience in order to further develop this approach. We expect to receive some large SUN workstations, which will permit us to perform larger experiments with FORLOG.

Task 3: Develop techniques for equality reasoning in the ATMS. Equality reasoning plays an important role in FORLOG programs, because it is used to infer the values of Skolem constants. Unfortunately, equality is handled very inefficiently by the ATMS. A straightforward implementation of equality using only the symmetric and transitive axioms allows many redundant proofs to be derived from any set of equalities. For example, from the knowledge that $sk_1 = sk_2$, $sk_2 = sk_3$, and $sk_3 = sk_4$, these axioms will permit four distinct proofs that $sk_1 = sk_4$. In general, for n equality assertions, there are $O(n!)$ generated proofs. The bad news doesn't end here, however. Each of these proofs is cached by the ATMS in the form of a dependency record. Whenever a change is made in a supporting assumption, the ATMS must traverse each of these $O(n!)$ dependencies to update the database. This is clearly unacceptable.

Traditional techniques for representing and reasoning with equality (and other congruence closures; e.g., Nelson & Oppen, 1980) are not applicable in the ATMS, because they assume that the database of assertions represents a single state or context. However, equality is such a strong relation that there should be some more efficient technique for integrating equality reasoning into the ATMS. The approach that we plan to take is to construct a separate module for reasoning about equality. Justifications from non-equality assertions will support equality assertions, and non-equality assertions will be supported by equality assertions, but no justification links will be maintained from one equality assertion to another. Instead, a special-purpose updating algorithm will be developed to maintain the equality database.

At the present time, we think we have an algorithm that visits each equality assertion only once when processing a new assertion or a change in support. Further work is required to prove it correct and implement it efficiently.

Task 4: Develop a control language for FORLOG. Unlike Prolog, FORLOG has been developed thus far as a purely declarative language. This means that there is no way to exercise control over the execution of the program. This is particularly true when the extended style of "programming with assertions" is employed. In that style, evensimple programs, such as sorting or finding the maximum element of an array, require $O(n^2)$ time or worse, because they perform redundant computations.

Currently, the only option available to the programmer is to write procedures in lisp that guide reasoning by selecting appropriate inference steps from FORLOG's internal agendas. Within the Prolog community, many approaches have been explored for providing control including control annotations (e.g., the cut, goal suspension annotations, etc.), meta-level logics, second-order logics, and so on. We plan to explore these methods and apply them to FORLOG. The goal will be to develop a control language that allows the programmer to state preferences concerning the contexts in which problem solving should occur without writing intricate or tricky procedures.

Our first goal will be to develop a language in which it is easy to express the dependency-directed backtracking procedure described by deKleer and Williams (1986). Then we will explore control issues that arise in the programs that are developed in task 2.

3 Learning by Experimentation—Thomas G. Dietterich

3.1 Introduction

An important goal of current machine learning research is to find techniques for strengthening the weak methods of inductive learning developed during the seventies (e.g., Quinlan, 1986; Michalski, 1983; Mitchell, 1982). There are essentially two avenues being pursued.

One is to provide the learning system with more background knowledge and develop ways in which that background knowledge can be exploited to learn new knowledge. Explanation-based learning (e.g., Mitchell, Keller & Kedar-Cabelli, 1986; DeJong & Mooney, 1986) is an example of this line of research.

The other avenue is to find ways in which the learning system can exploit the richness of the learning environment. Learning by experimentation (e.g., Dietterich & Buchanan, 1983; Rajamoney, DeJong & Faltings, 1985; Rajamoney, 1986) is pursuing this second line of research. The key hypothesis is that giving the learning system the ability to actively experiment with its environment will improve and extend its learning capabilities.

A few researchers have developed systems that learn by experimentation (e.g., Kibler and Porter, 1983; Shapiro, 1981, 1982; Mitchell, Utgoff & Banerji, 1983; and Lenat, 1980, 1983a, 1983b). Most of the research so far has focused on (a) demonstrating the power of experimentation and (b) developing methods for the design and execution of experiments. We believe the time has come to take a deeper look at the underlying issues.

3.2 Four Issues in Experimentation

Under a grant (IST-8519926) from NSF, we are exploring four fundamental issues in experimentation.

1. How does experimentation-based learning compare with learning from environment-selected training examples? It would appear that the key advantage of experimentation-based learning is that the learner can select the training example that would be most informative instead of waiting for the environment (or a teacher) to present such an example. On the other hand, the teacher knows the "right answer" and therefore can select training examples that can efficiently isolate the desired theory or concept. To date, no studies have investigated this issue in any depth.

2. How do *instance selection* experiments compare with *decomposition* experiments? Most machine learning work on experimentation has viewed it as simply a method for obtaining informative training examples. However, an alternative view is that experiments are actions that the learner takes in order to *simplify* the learning problem. The clearest case involves *in vitro* experimentation, where a scientist isolates a small subcomponent (e.g., an individual cell or an individual molecule) and studies its behavior in the test tube. We call such actions *decomposition* experiments. To our knowledge, no experimentation system has studied decomposition experiments.

3. What criteria should a learner apply to select the best experiment to perform? At each point in the learning process, the learner must choose the action to perform next. We have given the name *experiment bias* to the set of critera that the learner uses to make this decision. In principle, statistical decision theory can be applied to derive experiment bias from the more familiar *theory bias*—the criteria the learner applies to prefer one theory over another (Utgoff, 1986). Each theory can be assigned a prior probability (a form of theory bias) and then the experiment can be chosen that would be maximally informative in the sense that it would maximally reduce the uncertainty (entropy) of the learner concerning the identity of the correct theory. Important questions include: (a) How effective is the experiment bias that is derived from theory bias? (b) How important is experiment bias in general? Early in the learning process, it may not make much difference which experiment is selected. (c) If the experiment bias is wrong, how can it be corrected? Can randomly chosen experiments help?

Previous work has virtually ignored the question of experiment bias and its role in aiding or hindering learning. It is a difficult topic to study, because one must be able to *change* the experiment bias of the learning system and measure the resulting changes in learning performance.

4. How should the tradeoff between experimentation and theory formation be managed? A practical learning system cannot spend all of its time designing and performing experiments—some time must also be spent developing and refining theories that explain the data and provide guidance for future experiments. What is the proper mix of theory formation and experimentation? Significant computational as well as epistemological issues arise here.

Previous work has not provided much insight into this issue. Most existing experimentation systems implement only one point along the tradeoff between experimentation and theory formation. To study this question, we need to learn more about the tradeoff itself.

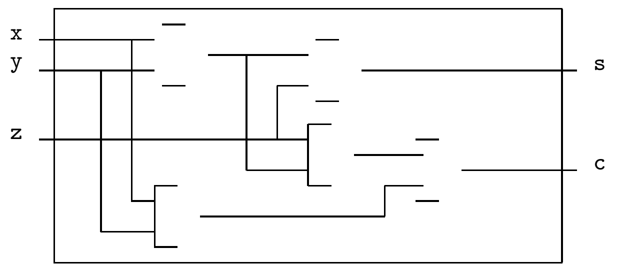

Figure 1: The combinational logic circuit of a full adder.

3.3 Current Research

To explore these issues, we are constructing a learning system for the simple domain of combinational (state-free) digital logic circuits. We plan to conduct a series of experiments on the learning system itself in order to understand learning by experimentation.

A combinational logic circuit can be viewed in many ways. First, it can be described structurally in terms of the circuit topology (see Figure 1). Second, it can be described functionally in terms of the truth table that it implements. Third, it can be described in more abstract functional terms as, for example, an adder, multiplexer, permutation network, etc. When an abstract functional representation is specified, it is also necessary to provide a mapping between the abstract data types, (e.g., numbers) and the boolean terminals. For example, the mapping for an adder might state that numbers are represented as 2's-complement binary strings.

In this domain, the goal of the learning system is to discover the structure and function of the circuit by performing experiments on it. Simple, instance-selection experiments involve placing desired boolean signals on the input terminals and observing the resulting values on the output terminals. More interesting decomposition experiments involve removing a subnetwork from the circuit and performing experiments on it in isolation. Our plan is to structure the circuit recursively as a network of modules. In an experiment, the learning system will be able to extract a randomly-chosen module from the larger circuit, but it will not know what this module was connected to (or any other information about the circuit topology). The space of theories in this domain involves three interrelated subspaces: the space of all structurally distinct circuits, the space of truth tables, and the space of abstract functions (with associated data type mappings).

To make it easy to explore the four issues listed above, the learning system for combinational logic circuits will use "brute force" techniques. By "brute force," we mean that the learning system will be given an enumeration of the space of theories (and experiments). This will allow us to change theory biases and experiment biases and compare the resulting learning behavior.

The space of theories that will be supplied to the program will be computed by Monte Carlo sampling of the space of all possible logic circuits. Random sampling is necessary because the space of interesting devices includes all circuits containing up to 22 nodes (inputs + outputs + internal gates). We estimate there are more than 10^{31} such circuits. We will employ exhaustive enumeration for all circuits with fewer than 9 nodes in order to calibrate the Monte Carlo procedures.

The brute force learning system will operate as follows. Each theory bias will be used to assign a prior probability to each theory. Whenever some new input data are obtained, the learning system will eliminate all theories that are inconsistent with the data and update the probabilities (according to Bayes' Rule) of the remaining theories. Different learning systems will be evaluated by comparing the maximum-likelihood predictions that they would make for unobserved data points. Different target circuits (that the learning systems are trying to understand) will be tested: ones that "obey" the theory bias, ones that do not, and ones selected at random.

Using this brute force learning system, we will be able to investigate each of the four issues listed above. First, we will compare instance-selection experiments with teacher-selected and randomly-selected experiments. Second, we will compare instance-selection experiments with decomposition (in vitro) experiments. We expect that this will show a significant advantage for decomposition experiments. However, it may be that a mix of the two approaches is best. Third, we will compare different kinds of experiment biases. By analyzing the table of possible theories, we can learn much about the relative value of biases over functional theories compared to biases over structural theories. It is also possible to have a bias only over the abstraction mappings. This information alone might provide significant guidance to the learning system.

The only issue that we will not be able to explore in depth concerns the tradeoff between theory formation and experimentation. We will determine how important the choice of particular experiments is at different points in the learning process. However, to fully explore this issue, we need to develop more sophisticated learning programs that generate their alternative theories and experiments incrementally, as needed. This will be a subject for future studies.

***

4 A Hybrid Approach to Reasoning Under Uncertainty—Bruce D'Ambrosio

4.1 Background

A complete approach to reasoning under uncertainty requires support for both identification of the appropriate hypothesis space and ranking hypotheses based on available evidence. We are developing a hybrid reasoning scheme which combines symbolic and numeric methods for uncertainty management to provide efficient and effective support for both of these tasks. The hybrid is based on symbolic techniques adapted from Assumption-based Truth Maintenance systems (ATMS), combined with numeric methods adapted from the Dempster/Shafer theory of evidence, as extended in Baldwin's Support Logic Programming system. The hybridization is achieved by viewing an ATMS as a symbolic algebra system for uncertainty calculations. This technique has several advantages over conventional methods for performing inference with numeric certainty estimates in addition to dynamic determination of hypothesis spaces, including improved management of dependent and partially independent evidence, faster run-time evaluation of propositional certainties, and the ability to query the certainty value of a proposition from multiple perspectives. In the remainder of this description we first briefly review research on numeric and symbolic approaches to reasoning under uncertainty, and then discuss our hybrid approach.

Current techniques for performing inference with numeric certainty values in expert systems (Buchanan and Shortliffe, 84), (Lowrance, 86), (Szolovits and Pauker, 78), (Weiss *et al*, 78) assume that the hypothesis space has been pre-determined, and rely on numeric combination of evidence at each stage of inference. All known evidence is reduced to a single number or pair of numbers associated with each proposition. This means that the source of each derived support is lost. This, combined with a lack of information regarding the necessary conditional probabilities, requires that independence assumptions be made in order to combine evidence from multiple inferences supporting a single proposition (Cheeseman, 85). Further, it is not possible to determine the sensitivity of consequent certainty values to individual pieces of evidence without re-doing the entire inference process. Finally, the simple numeric schemes typically used in expert systems usually do not allow for changes in support for antecedents once inference has taken place. Recent research (Pearl, 87; Shachter, 87) has been aimed at permitting direct expression of relationships and elimination of independence assumptions. This work, however, still assumes a pre-determined hypothesis space. In addition, the computational complexities of these approaches present difficulties, and the explanatory capabilities of the simple rule-based paradigm have not yet been reproduced.

An alternative paradigm for reasoning under uncertainty (Cohen, 84), (deKleer, 86a), (Doyle, 79), (Liu, 86), (McAllester, 80) can be traced back to the work of Doyle (Doyle, 79) on symbolic truth maintenance systems. One of the most recent of these systems is the Assumption-based Truth Maintenance System (ATMS) of deKleer (deKleer, 86a,b,c). This system, like many other symbolic systems, has the capability of dynamically determining hypothesis spaces. In the case of an ATMS, this can be done through the mechanism of *interpretation construction.* An ATMS builds symbolic expressions (*labels*) for the truth value of all propositions in a database. Specifically, the label of a proposition is a list of sets of assumptions, where the proposition is held to be true if all of the assumptions in at least one of the sets are true. This *structured* representation for uncertainty permits the algorithms to automatically eliminate dependent evidence and provides for sensitivity analysis of resulting truth values. However, it suffers from two limitations. First, the only ranking possible is between propositional truth values of false, true, and uncertain. This is inadequate for applications in which it is necessary to rank uncertain alternatives. Second, the computational complexity of the underlying algorithms is as unattractive as that of the newer numeric uncertainty schemes.

4.2 Research Description

We are engaged in research (D'Ambrosio, 87a,b,c) on a novel method for computing the certainty of derived propositions from numeric certainty information for the initial evidence, based on the mechanisms of the ATMS and the evidential combination algorithms of the Dempster/Shafer theory of evidence (Shafer, 76). The method relies on the propagation mechanisms in the ATMS to perform most evidential inference and combination operations symbolically, and only substitutes numeric values when asked for the certainty of a proposition. Advantages include improved handling of non-independent evidence, very fast run-time evaluation of propositional certainties, and the ability to request the certainty value of any proposition from several perspectives. Also, changes in support for antecedents are automatically reflected in consequent propositions, even after inference has occurred.

One view we can take of an ATMS is as a symbolic algebra system for uncertainty information. That is, an ATMS starts with a set of uncertain data and a set of inferences which can be drawn from that data. It then computes a closed form symbolic expression for the truth of all consequent propositions, in terms of the symbolic truth values (*assumptions*) of the original data. One consequence of this capability is that, once problem solving is complete, the truth of a proposition in any solution or partial solution can readily be determined with inexpensive, purely local operations.

If we view ATMS assumptions as certainty variables, and if we can establish a mapping between the computations performed by the ATMS and those of some well-founded numeric certainty calculus, then we can substantially improve its ability to rank alternatives. We are establishing such a mapping for the Dempster/Shafer theory of evidence, and a simplification/extension of it, Support Logic Programming (Baldwin, 85). The elements that must be mapped are basic mass assignments, primitive hypothesis sets, frames of discernment, inference rules (and associated algorithms for propagating evidence across inferences), and evidence combination algorithms.

We use an ATMS assumption to carry each element of a basic mass assignment. In the Dempster/Shafer theory of evidence, a source provides evidence not for a single proposition, but rather distributes evidential mass over an entire *frame of discernment*, the power set of a set of mutually exclusive and exhaustive alternatives (the *primitive hypothesis set*). We use the ATMS One-of disjunction to represent a basic mass assignment, and the mutual exclusion capabilities of an ATMS to construct a representation for a frame of discernment. Dempster/Shafer theory does not include a model for inference rules; however, SLP does, and the SLP inference rule

```
B :- A : [Sl(B|A), Su(B|A)]
```

can be mapped into two ATMS justifications, one from A to B supported by $Sl(B|A)$, and one from A to (not B) supported by $(1 - Su(B|A))$. The basic model of inference underlying this translation is that each inference rule is a rule specifying how to map evidence for elements in some set of frames of discernment (those containing the elements of A, assuming A is composite) into evidence over the frame of discernment containing B. That is, an inference rule generates a basic mass assignment. An important feature of this mapping is that the composition of a sequence of two justifications is parenthesis free. As a result, we can defer numeric evaluation, use the standard ATMS assumption propagation algorithm, and compute the support pair for any proposition by examining only the labels of propositions in its frame of discernment.

Dempster/Shafer theory provides a mechanism for combining sets of evidence over a frame of discernment consisting of a primitive set of exhaustive and mutually exclusive hypotheses. We have developed a hybrid symbolic/numeric equivalent of this mechanism which performs as much of this process as possible symbolically, and defers the final numeric computations untill the system is actually queried for the certainty value for a specific proposition. The process consists of the following steps:

- Symbolic conversion of mass functions to belief functions.
- Symbolic expansion of labels to eliminate duplicate or inconsistent evidence components.
- Reduction of the expanded symbolic belief expression to a numeric value.
- Computation of Kappa - the Dempster/Shafer normalization parameter.
- Normalization of the numeric belief value.

This process is repeated for the negation of the proposition, yielding the belief in the negation, from which the plausibility of the proposition can be easily recovered[2].

Currently, we only support subsets of the frame of discernment consisting of primitive hypotheses and their negations. This subset has been adequate for the problems we have addressed, and has the advantage that computational complexity of evidence combination is linear in the number of primitive hypotheses. We are also developing symbolic techniques for representing the full power set of the primitive hypothesis set, as well as hierarchical subsets of the power set.

Notice that we can now perform a sensitivity analysis on propositional support. This can be done by selecting assumption sets to be included in the support computation, or by substituting extreme values for the numeric support for a particular assumption prior to evaluation. One method of selecting assumption sets is to specify a kernel set of assumptions. An assumption set can then be included in the support computation according to various measures of its consistency with the kernel set.

[2]In Dempster/Shafer theory, the certainty of a hypothesis is represented as a pair (Belief, Plausibility), where $Plausibility(X) = 1 - Belief((NotX))$.

4.3 Tasks

We have already constructed a partial prototype of the ideas presented here. However, much remains to be done in design, prototype construction, and evaluation. We plan to extend and refine our previous work by developing techniques for symbolically representing the full power set of the primitive hypothesis set, as well as hierarchical subsets of it. We need to develop a better understanding of the model of inference we are using, and develop a formal characterization of it. We plan to develop a complete prototype reasoning facility incorporating all of the techniques we have described, and evaluate it in several significant reasoning tasks, yet to be selected.

Other Numeric Uncertainty Models We have shown that deKleer's ATMS can be used as a symbolic algebra system for uncertainty calculations consistent with at least one uncertainty calculus. We have not established, though, how wide a class of numeric certainty algorithms can be captured using this approach. One criteria we did identify is that the mapping to the ATMS must be such that the calculations are associative, since ordering and nesting information is lost in assumption-set propagation. We believe that a Bayesian model, as well as fuzzy logic (Zadeh, 79), can be adapted to representation symbolically in the ATMS, and are beginning to develop mappings for each.

Viewing Numeric Certainties as Summaries We mentioned briefly above that neither the symbolic nor the newer numeric approaches to uncertainty management scaled well. We believe the hybrid approach we have presented offers a solution to this problem. The above research is predicated on a view in which numeric certainty values are taken as primitive, and ATMS assumptions are merely truth variables. This view permits us to rectify a fundamental limitation of the ATMS, the inability to rank alternatives. However, ATMS algorithms compute the consequences of any change (addition of a proposition, justification, or assumption) for every proposition in the database. The expense of this computation grows rapidly with the number of propositions and assumptions being managed.

The fundamental cause of this problem is the same feature of the ATMS that provides its richness and power, namely that certainty in a proposition is represented by a complex, structured set of tokens. One way to solve this problem is to *summarize* the certainty in a more compact form, and relax the commitment to immediately compute all consequences of a change in support for every proposition. Many problems can be partitioned into a series of steps, where each step can be viewed as a cycle of alternative development followed by selection.

We plan to address the complexity problem by partitioning the ATMS into separate databases for each reasoning task. Inferences in one partition dependent on hypotheses located in another partition will be linked by creating an *assumption* in the dependent partition. The assumption will be tagged with the numeric support value obtained from the hypothesis in the antecedent partition. A forward link from the hypothesis to the new assumption can also be kept, so that the assumption can be informed of *significant* changes in support for the antecedent proposition. This propagation between partitions, however, is not done automatically by the ATMS, but rather by the higher-level problem-solver.

Search Control The availability of numeric evaluations of alternatives significantly impacts the architecture of problem-solvers built using an ATMS. As discussed earlier, an ATMS is an elegant mechanism for exploring a set of alternatives, but provides limited mechanisms for ranking them. In part for this reason, previous work using an ATMS has stressed *completeness.* Alternatives are explored whenever there is any evidence that an alternative is possible (i.e., when the label for the corresponding proposition is non-null). This is the basic strategy available in the prototype rule-based problem-solver we have implemented, as well as the one used in the commercially available tool, ART. With numeric estimates, at least two alternate heuristic search strategies suggest themselves:

1. Pursue the currently highest ranked alternative(s).
2. Pursue all alternatives with rankings above some (perhaps dynamically adjustable) threshold.

4.4 Summary

Uncertainty is a pervasive problem underlying much of AI. We rarely have complete information, and even when we do, we can only afford to apply it in its entirety in toy problems. Numeric methods for managing uncertainty provide a means of ranking alternatives, but do not directly support their development. Symbolic methods directly support alternative development, but provide little guidance in selection. By combining these two classes of techniques, we are developing a system that directly supports both aspects of reasoning under uncertainty and does so in an efficient and effective fashion.

5 Problem-Solving with Multiple Qualitative Models — Bruce D'Ambrosio

5.1 Introduction

Qualitative modelling (Bobrow, 1984) is vital to many important applications of Artificial Intelligence, including diagnosis, design, repair, and real-time control (Raulefs, D'Ambrosio, Fehling, *et al*, 1987). In previous research we have identified the necessity for, and sketched a theory of, problem-solving via coordinated application of multiple models of a physical situation (D'Ambrosio, 1985; D'Ambrosio & Fehling, 1985). We are currently engaged in research to improve the foundations of this theory, develop a research prototype for experiments intended to test and generalize it, and develop methods within the theory for coordinated application of multiple models to specific reasoning tasks. Such methods are currently unavailable and are essential to the construction of efficient and effective problem solvers utilizing model-based reasoning.

5.2 Background

Research to date on model-based reasoning has largely been concerned with demonstrating the existence of adequate representations and reasoning methods for single models (deKleer & Brown, 1984; Kuipers, 1986). Application of this research has usually resulted in the implicit adoption of the *Universal Model hypothesis* (UMH) - that a single global theory at a single grain size is adequate for all problem solving. Human problem solvers, however, seem adept at selecting a perspective from which an efficient and powerful model of a problem can be formulated, and at dynamically modifying that perspective as problems arise. Thus, current research efforts are far from capturing the full range of human model-based reasoning. We believe that complex problem solving requires the coordinated application of a variety of models of a physical situation. We have termed this the *Multiple Model hypothesis* (MMH).

5.3 The Multiple Model Hypothesis

The MMH is based, in part, on a view of human application of domain knowledge to physical situations, described in (D'Ambrosio, 1986), consisting of the following tasks:

1. Selection of a theoretical perspective from which to view the situation (e.g., heat-flow or ballistics).
2. Segmentation of the situation into components, instances of the types provided by the theory.
3. Selection of a mathematics in which to construct a formal model of the situation.
4. Formal model construction.
5. Transformation of the formal model (i.e., solution).
6. Interpretation of the results of formal model transformation, including evaluation of their applicability to the original situation.

The Multiple Model hypothesis arises in response to multiple or changing goals, as well as dynamically created subgoals. For example, consider a simple monitoring and assessment task: (a) a model used to generate norms for monitoring may not be detailed enough to allow accurate fault isolation (multiple goals), (b) the model initially selected for fault isolation may become inappropriate when new data indicates a more rapid response is required than it can provide (changing goals), or (c) a model of one aspect of a system may require an approximate solution to some other aspect (dynamic subgoals).

We believe there are two critical problems for multiple-model reasoning: (a) identification of the control principles behind such intelligent coordinated model application and (b) development computational frameworks in which these principles can be implemented. We discuss each of these below.

Identifying Control Principles Surprising little research has been done within the AI community on selection and control of problem solving models (Amarel, 1986). Davis (Davis, 1984) has suggested solutions to specific problems, but no general solution framework has emerged. We are adopting a three phase approach to studying these problems. First, we are beginning a thorough review of the psychological literature on physical problem solving (e.g., (Gentner & Stevens, 1983)). Second, we intend to use the techniques of protocol analysis (Ericsson & Simon, 1980) to identify the heuristics used by human experts on actual problems in a selected domain. Third, our long range goal is to develop a set of first principles based on decision theoretic criteria from which these heuristics can be derived. This aspect of our research makes relatively light equipment demands, and is not further detailed here.

A Testbed for Multiple Model Reasoning We are developing a testbed facility designed to support our view of model-based reasoning and to permit experimentation to test and refine this view. The architecture of this facility is based on the following hypotheses:

- A model is an *active problem-solving element*, consisting of not only a representation of the system to be reasoned about, but also algorithms for reasoning with that representation and control principles for organizing model activity. Object-oriented programming provides for encapsulation of the representation and reasoning aspects of a model, but in its pure form (ala

Smalltalk) does not provide a suitably rich representation for control. The blackboard architecture provides a paradigm for describing external control of a model, but does not provide mechanisms for internal control. The Schemer (Raulefs, D'Ambrosio, Fehling, *et al*, 1986) architecture, combines and extends both paradigms to provide an appropriate framework for model-based reasoning under the multiple model hypothesis. It provides the necessary encapsulation and abstract typing facilities, together with representations for both external and internal control.

- While reasoning under the universal model hypothesis requires only intra-model search control and consistency management, reasoning under the multiple model hypothesis requires both intra-model and inter-model search control and consistency management, as well as evaluation facilities. The symbolic truth maintenance techniques, particularly the ATMS (deKleer, 1986a), provide a sufficient starting point for intra-model search control and consistency management. These techniques can be generalized to yield a mechanism adequate for inter-model management as well. We have sketched a mechanism for partitioning an ATMS database, using numeric certainty estimates as label summaries and employing problem-solver control of communication between partitions.

- Model determination, application, and evaluation depend heavily on heuristic control decisions. While symbolic truth maintenance systems such as the ATMS provide elegant and efficient mechanisms for alternative elaboration, they provide very limited support for alternative selection and decision making. In previous work (D'Ambrosio, 1987a; see also previous section) we have begun to integrate well-founded numeric methods for reasoning with uncertain information into an ATMS, thus providing direct support for alternative selection, and potentially making available the powerful tools developed by the decision analysis community.

6 Bibliography

Amarel, S. 1986. Problem solving. Rep. No. DCS-TR-188. Department of Computer Science, Rutgers University, New Brunswick, NJ.

Baldwin, J. 1985. *Support Logic Programming.* Technical Report ITRC 65, Information Technology Research Center, Univ. of Bristol.

Barnet, J. 1981. Computational Methods for a Mathematical Theory of Evidence. In *Proc. of IJCAI-81*, 868–875.

Bobrow, D. 1985. (Editor) *Qualitative Reasoning about Physical Systems,* Cambridge, MA: MIT Press.

Buchanan, B. and Shortliffe, E. 1984. *Rule-Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley.

Cheeseman, P. 1985. In Defense of Probability. In *Proceedings of IJCAI85*, Los Altos, CA: Morgan-Kaufmann. 1002–1009.

Clark, K. L. 1978. Negation as failure. In *Logic and Databases*, H. Gallaire and J. Minker (Eds.) New York: Plenum Press. 293–322.

Clocksin, W. F., and Mellish, C. S. 1984. *Programming in Prolog*, Berlin: Springer-Verlag.

Cohen, P. 1984. *Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach.* Pitman, London.

D'Ambrosio, B., 1985. Levels of abstraction in model-based reasoning. In *Proceedings of Robexs 85*, pages 19–22, NASA/Johnson Space Center, June 1985.

D'Ambrosio, B., 1986. *Qualitative Process Theory using Linguistic Variables.* Ph.D. thesis, University of California, Berkeley.

D'Ambrosio, B. 1987a. Truth Maintenance with Numeric Certainty Estimates. In *Proceedings of IEEE AI Applications Conference*, IEEE Computer Society Press, Los Angeles. 244–249.

D'Ambrosio, B. Enriching the Mathematics in Qualitative Process Theory. To appear in the Proceedings of AAAI-87 (Seattle, Wash. July 12-16).

D'Ambrosio, B. and Fehling, M. R., 1985. Selection and construction of constraint-based models. In *SIGART Newsletter*, July, 1985.

Davis, R., 1984. Diagnostic reasoning based on structure and behavior, *Artificial Intelligence*, 24(1-3):347-410,December 1984.

deKleer, J. 1986a. An assumption-based TMS, *Artificial Intelligence*, 28(2) 127–162.

deKleer, J. 1986b. Extending the ATMS. *Artificial Intelligence.* 28(2): 163–196.

deKleer, J. 1986c. Problem-solving with the ATMS, *Artificial Intelligence*, 28(2) 197–224.

deKleer, J. and Brown, J. S., 1984. A qualitative physics based on confluences. *Artificial Intelligence*, 24(1-3):7–83, December 1984.

deKleer, J., and Williams, B. C. 1986. Back to backtracking: controlling the ATMS. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, Philadelphia, PA. Los Altos, CA: Morgan-Kaufmann, 910–917.

DeJong, G., Mooney, R., 1986. Explanation-based learning: an alternative view. *Machine Learning*, 1 (2) 145–176.

Dietterich, T. G., and Buchanan, B. G., 1983. The role of experimentation in theory formation. In *Proceedings of the International Machine Learning Workshop*, Allerton House, Monticello, IL. 147–155.

Doyle, J. 1979. A Truth Maintenance System. *Artificial Intelligence.* 12(3): 231–272.

Duda, R., Hart, P., Nilson, N. 1976. Subjective Bayesian Methods for Rule-Based Inference Systems. In *Proceedings of 1976 National Computer Conference*, 1075–1082, AFIPS.

Ericsson, K. A. and Simon, H. A. 1980. Verbal reports as data. *Psychological Review*, 87 (3) 215–251.

Fickas, S. 1985. Automating the Transformational Development of Software, *IEEE Transactions on Software Engineering,* Vol. SE-11 (11). 1268–1277.

Flann, N. S., Dietterich, T. G., and Corpron, D. R., 1987. Forward chaining logic programming with the ATMS. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-87)*, Seattle, WA. Los Altos, CA: Morgan-Kaufmann.

Forgy, C. L., 1982. RETE: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19 (1) 17–37.

Gentner, D. and Stevens, A., 1983. *Mental Models*, Lawrence Erlbaum, New Jersey.

Gordon, J. and Shortliffe, E. 1985. A Method for Managing Evidential Reasoning in a Hierarchical Hypothesis Space. *Artificial Intelligence* 26(3): 323–358.

Green, C., Luckham, D., Balzer, R. Cheatham, T., and Rich, C. 1983. *Report on a Knowledge-Based Software Assistant.* Technical Report KES.U.83.2, Kestrel Institute.

Kibler, D., and Porter, B., Perturbation: A means for guiding generalization, *Proceedings of IJCAI-83*, 415-418, Karlsruhe, Germany, 1983.

Kuipers, B. J., 1986. Qualitative simulation. *Artificial Intelligence*, 29(3):289–338, September 1986.

Lenat, D. B., AM: An artificial intelligence approach to discovery in mathematics as heuristic search, In Davis, R., and Lenat, D. B., *Knowledge-based systems in Artificial Intelligence*, 1980.

Lenat, D. B., Theory formation by heuristic search, The Nature of Heuristics II: Background and Examples, *Artificial Intelligence*, Vol. 21, 31-60, 1983a.

Lenat, D. B., EURISKO: A program that learns new heuristics and domain concepts, The Nature of Heuristics III: Program Design and Results, *Artificial Intelligence*, Vol. 21, 61–98, 1983.

Liu, G. 1986. Causal and Plausible Reasoning in Expert Systems. In *Proceedings of AAAI86*, 222–225. AAAI.

Lowrance, J. 1986. A Framework for Evidential-Reasoning Systems. In *Proceedings of AAAI86*, 896–903. AAAI.

McAllester, D. 1980. *An Outlook on Truth Maintenance.* AI Memo 551, MIT AI Lab.

Michalski, R. S., 1983. A theory and methodology of inductive learning. *Artificial Intelligence*, 20 (2) 111–161.

Mitchell, T. M., 1982. Generalization as search. *Artificial Intelligence*, 18 (2) 203–226.

Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T., 1986. Explanation-based generalization: a unifying view. *Machine Learning*, 1 (1) 47–80.

Mitchell, T., Steinberg, L, Shulman, J., 1984. VEXED: a knowledge-based VLSI design consultant, Rutgers AI/VLSI Project Working Paper No. 17.

Mitchell, T. M., Utgoff, P. E., and Banerji, R. B., Learning by experimentation: acquiring and refining problem-solving heuristics, In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., (eds.), *Machine Learning*, Palo Alto: Tioga, 1983.

Neches, R. Swartout, W. Moore, J. 1985. Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions of Software Engineering*, SE-11 (11):1337–1350.

Nelson, G., Oppen, D. C., 1980. Fast decision procedures based on congruence closures. *J. ACM* 27 (2), 356–364.

Partsch, H. and Steinbruggen, R. 1983. Program transformation systems. *Computing Surveys,* 15(3):199–236.

Pearl, J. 1986. On the Logic of Probabilistic Dependencies. In *Proceedings of AAAI-86*, Los Altos, CA: Morgan-Kaufmann. 339–343.

Quinlan, J. R., 1986. Induction of decision trees. *Machine Learning*, 1 (1) 81–106.

Rajamoney, S., DeJong, G. F., and Faltings, B., 1985. Towards a model of conceptual knowledge acquisition through directed experimentation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, Los Angeles, CA. 688-690.

Rajamoney, S. 1986. Automated design of experiments for refining theories. Report Number UILU-ENG-86-2213, Coordinated Science Laboratory, University of Illinois, Urbana, IL.

Raulefs, P., D'Ambrosio, B., Fehling, M., et al., 1987. Real-time Process Management for Materials Composition. In *Proceedings Third Conference on AI Applications*, pages 120–125, Kissimmee, Florida, Computer Society of the IEEE, February, 1987.

Shafer, G. *A Mathematical Theory of Evidence.* Princeton University Press, 1976.

Shapiro, E. Y., Inductive Inference of Theories from Facts, Rep. No. 192, Department of Computer Science, Yale University, 1981.

Shapiro, E. Y., Algorithmic Program Debugging, (Doctoral dissertation) Rep. No. 237, Department of Computer Science, Yale University, 1982.

Szolovits, P. and Pauker, S. 1978. Categorical and Probabilistic Reasoning in Medical Diagnosis. In *Artificial Intelligence*, 11: 115–144.

Utgoff, P. E., 1986. Shift of bias for inductive concept learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.) *Machine Learning* Volume 2, 107–148.

Warren, D. H. D., 1983. An abstract prolog instruction set. Technical Note 309, Menlo Park, CA: SRI International.

Weiss, S., Kulikowski, C., et al. 1978. A Model-based Method for Computer-Aided Medical Decision Making. In *Artificial Intelligence*, 11:145–172.

Wile, D. S. 1983. Program developments: Formal explanations of implementations. *Communications of the ACM*, 26(11):902–911.

Zadeh, L. 1979. A Theory of Approximate Reasoning. In *Machine Intelligence 9*, John Wiley and Sons.

7 Recent Publications in AI from Oregon State University

Corpron, D. R. *Disjunctions in forward-chaining logic programming.* Technical Report 87-30-1. Department of Computer Science, Oregon State University, Corvallis, OR.

D'Ambrosio, B., 1985. Levels of abstraction in model-based reasoning. In *Proceedings of Robexs 85*, pages 19–22, NASA/Johnson Space Center, June 1985.

D'Ambrosio, B., 1986. *Qualitative Process Theory using Linguistic Variables.* Ph.D. thesis, University of California, Berkeley.

D'Ambrosio, B. 1987a. Truth Maintenance with Numeric Certainty Estimates. In *Proceedings of IEEE AI Applications Conference*, IEEE Computer Society Press, Los Angeles. 244–249.

D'Ambrosio, B. Enriching the Mathematics in Qualitative Process Theory. *Proceedings of AAAI-87* (Seattle, Wash. July 12-16).

D'Ambrosio, B. and Fehling, M. R., 1985. Selection and construction of constraint-based models. In *SIGART Newsletter*, July, 1985.

Dietterich, T. G., (1986). Learning at the knowledge level, *Machine Learning*, 1(3) 287–316.

Dietterich, T. G., (1986). Induction: Weak but Essential (commentary on Schank, Collins, and Hunter), *Behavioral and Brain Sciences*, 9 (4), 1986, 654–655.

Dietterich, T. G., (1986). Proceedings of the Workshop on Knowledge Compilation (editor), Technical report, Department of Computer Science, Oregon State University, Corvallis, OR.

Dietterich, T. G. In press. A knowledge level analysis of learning programs. In Michalski, R. S. and Kodratoff, Y. (Eds.) *Machine Learning: An artificial intelligence approach,* Vol. III. Los Altos, CA: Morgan-Kaufmann.

Dietterich, T. G., and Bennett, J. S., The Test Incorporation Hypothesis and the Weak Methods, Rep. No. TR 86-30-4, Department of Computer Science, Oregon State University, Corvallis, OR.

Dietterich, T. G., and Bennett, J. S., The Test Incorporation Theory of Problem Solving, in T. G. Dietterich (ed.) *Proceedings of the Workshop on Knowledge Compilation*, Otter Rock, OR, September 24–26, 1986. Technical Report. Department of Computer Science, Oregon State University, Corvallis, OR 97331.

Dietterich, T. G., Flann, N. S., and Wilkins, D. C., (1986). A Summary of Machine Learning Papers from IJCAI-85, *Machine Learning*, 1 (2), 227–242.

Dietterich, T. G., and Michalski, R. S., 1985. Discovering patterns in sequences of events, *Artificial Intelligence* (25), 187–232.

Dietterich, T. G., and Ullman, D. G., 1987. FORLOG: A Logic-Based Architecture for Design, in *Expert Systems in Computer-Aided Design*, North-Holland, and presented at IFIP WG5.2 Working Conference on Expert Systems in Computer-Aided Design, Sydney, Australia, February, 1987.

Flann, N. S., (1986). Learning Functional Descriptions from Examples, Masters Thesis, Department of Computer Science, Oregon State University, Corvallis, OR.

Flann, N. and Dietterich, T. G., (1986). Selecting appropriate representations for learning from examples. In *Proceedings of the National Conference on Artificial Intelligence: AAAI86*, Philadelphia, PA. Los Altos, CA: Morgan-Kaufmann, 460–466.

Flann, N. S., Dietterich, T. G., and Corpron, D. R., 1987. Forward chaining logic programming with the ATMS. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-87)*, Seattle, WA. Los Altos, CA: Morgan-Kaufmann.

Prieditis, A. E., Dietterich, T. G., Hirsh, H., Kedar-Cabelli, S. T., Kempinski, R. V., Minton, S., and Subramanian, D., (1987). AAAI-86 Learning Papers: Developments and Summaries. *Machine Learning* 2(1) 83–96.

Ullman, D. G., and Dietterich, T. G., (In press). Mechanical Design Methodology: Implications on Future Developments of Computer-Aided Design and Knowledge-Based Systems, To appear in *Engineering with Computers.*

Ullman, D. G., Stauffer, L. A., Dietterich, T. G. (In press). Preliminary results of an experimental study of the mechanical design process.